

# Package ‘rsprng’

February 15, 2012

**Version** 1.0

**Date** 2010-03-01

**Title** R interface to SPRNG (Scalable Parallel Random Number Generators)

**Author** Na (Michael) Li <wuolong@gmail.com>

**Maintainer** Na (Michael) Li <wuolong@gmail.com>

**Depends** R (>= 1.2.0)

**Description** Provides interface to SPRNG 2.0 APIs, and examples and documentation for its use.

**License** GPL (>= 2)

**URL** <http://www.r-project.org/>

**Repository** CRAN

**Date/Publication** 2010-03-09 07:40:55

## R topics documented:

free.sprng . . . . .	2
init.sprng . . . . .	3
pack.sprng . . . . .	6
spawn.new.sprng . . . . .	7
spawn.sprng . . . . .	8
type.sprng . . . . .	9
unpack.sprng . . . . .	10

<b>Index</b>	<b>11</b>
--------------	-----------

---

`free.sprng`*Free memory used by SPRNG*

---

### Description

`free.sprng` frees the memory allocated to save the PRNG states by SPRNG and calls `RNGkind` to restore R's default RNG.

### Usage

```
free.sprng ()
```

### Details

SPRNG allocates some memory to save the state of the underlying RNG and the memory must be explicitly freed.

### Value

Return current PRNG state and NULL if there were no active PRNG stream.

### Author(s)

Na (Michael) Li <nali@umn.edu>

### References

SPRNG: Scalable Parallel Random Number Generator Library Web Page. <http://sprng.cs.fsu.edu/>

### See Also

[init.sprng](#), [pack.sprng](#)

### Examples

```
init.sprng (1, 0, seed = 231, kind = "PMLCG")
runif (10)
free.sprng ()
```

---

init.sprng	<i>Initialize SPRNG (Scalable Parallel Random Number Generator)</i>
------------	---

---

### Description

init.sprng initializes the parallel random number generator types and seeds.

RNGkind ("user") is called if necessary so that the default RNG is substituted with the parallel one.

### Usage

```
init.sprng (nstream, streamno, seed = 0, kindprng = "default", para = 0)
```

### Arguments

nstream	total number of random number streams to be initialized
streamno	the rank of the the random number stream in current process (valid value is from 0 to nstream - 1)
seed	an integer of random number seed. It is not the starting state of the sequence; rather, it is an encoding of the starting state. The same seed for all the streams. Distinct streams are returned. Only the 31 least significant bits of seed are used in determining the initial starting state of the stream. Default value is 0 where the default seed for each random number generator is used
kindprng	a character string of the disired kind of parallel random number generator
para	additional parameters for the parallel random number generators. If para is 0, default parameters for each PRNG are used. When invalid parameter is given, a warning is issued and the default paramter is used.

### Details

The currently available PRNG kinds are given below. prngkind is partially matched to this list. The default is "LFG".

1. "LFG" *Modified Lagged Fibonacci Generator*

The recurrence relation for this sequence of random numbers is given by the following equation:

$$z_n = x_n \text{XOR} y_n$$

where XOR is the exclusive-or operator,  $x$  and  $y$  are sequences obtained from Lagged Fibonacci sequences  $X$  and  $Y$  of the following form:

$$X_n = (X_{n-k} + X_{n-l}) \text{Mod} M$$

$$Y_n = (Y_{n-k} + Y_{n-l}) \text{Mod} M$$

$l$  and  $k$  are called the lags of the generator, and we use the convention that  $l > k$ .  $M$  is chosen to be  $2^{32}$ .  $X_n$  and  $Y_n$  are 32 bit integers.  $x$  is obtained from  $X$  by setting the Least Significant Bit of the latter to 0.  $y$  is obtained from  $Y$  by shifting the latter right by one bit. This modification of the Lagged Fibonacci Generator is performed in order to avoid certain correlations that are observed in the unmodified generator.

The period of this generator is  $2^{31}(2^l - 1)$  where  $l$  is the lag. For the default generator with lag  $l = 1279$ , the period is approximately  $2^{1310}$ . The number of distinct streams available is  $2^{31(l-1)-1}$ . For the default generator this gives  $2^{39648}$  distinct streams.

The parameters to this generator are the values of the lags. Allowed para values are:

para	l	k
0 (default)	1279	861
1	17	5
2	31	6
3	55	24
4	63	31
5	127	97
6	521	353
7	521	168
8	607	334
9	607	273
10	1279	418

2. "LCG" 48 Bit Linear Congruential Generator with Prime Addend

The recurrence relation for the sequence of random numbers produced by this generator is given by the following recurrence:

$$x_n = (ax_{n-1} + p) \text{Mod} M$$

where  $x_n$  is the  $n$  th term in the sequence,  $p$  is a prime number and  $a$  is the multiplier. The value of  $M$  for this generator is  $2^{48}$ . Different random number streams are obtained by choosing different prime numbers as the addend  $p$ . The period of this generator is  $2^{48}$ . The number of distinct streams available is of the order of  $2^{19}$ .

The multiplier  $a$  is a parameter to this generator. Allowed para values are 0 to 6, corresponding to 7 predefined multipliers.

3. "LCG64" 64 Bit Linear Congruential Generator with Prime Addend

The features of this generator are similar to the "LCG", except that the arithmetic is modulo  $2^{64}$ . The multipliers and prime addends  $p$  for this generator are different from those for the 48 bit generator.

The period of this generator is  $2^{64}$ . The number of distinct streams available is over  $10^8$ . Allowed para values are 0 to 2, corresponding to 3 predefined multipliers.

4. "CMRG" Combined Multiple Recursive Generator

This generator is defined by the following relation:

$$z_n = (x_n + y_n * 2^{32}) \text{Mod} 2^{64}$$

where  $x_n$  is the sequence generated by the 64 bit Linear Congruential Generator and  $y_n$  is the sequence generated by the following prime modulus Multiple Recursive Generator:

$$y_n = (107374182 * y_{n-1} + 104480 * y_{n-5}) \text{Mod} 2147483647$$

The same prime modulus generator is used for all the streams. Streams differ due to differences in the 64 bit LCG. The period of this generator is around  $2^{219}$ . The number of distinct streams available is over  $10^8$ .

The multiplier  $a$  for the 64 bit LCG is a parameter to this generator. Allowed para values are 0 to 2, corresponding to 3 prefine multipliers (same as "LCG64").

#### 5. "MLFG" *Multiplicative Lagged Fibonacci Generator*

The recurrence relation for this sequence of random numbers is given by the following equation:

$$x_n = (x_{n-k} * x_{n-l}) \text{Mod} M$$

$l$  and  $k$  are called the lags of the generator, and we use the convention that  $l > k$ .  $M$  is chosen to be  $2^{64}$ .

The period of this generator is  $2^{61}(2^l - 1)$  where  $l$  is the lag. For the default generator with lag  $l = 17$ , the period is approximately  $2^{81}$ . The number of distinct streams available is  $2^{63(l-1)-1}$ . For the default generator this gives around  $2^{1008}$  distinct streams.

The parameters to this generator are the values of the lags. Allowed para values are:

para	l	k
0 (default)	17	5
1	31	6
2	55	24
3	63	31
4	127	97
5	521	353
6	521	168
7	607	334
8	607	273
9	1279	418
10	1279	861

#### 6. "PMLCG" *Prime Modulus Linear Congruential Generator*

This generator is defined by the following relation:

$$x_n = (ax_{n-1}) \text{Mod} 2^{61} - 1$$

where the multiplier  $a$  differs for each stream. The multiplier is chosen to be certain powers of 37 that give maximal period cycles of acceptable quality.

The period of this generator is  $2^{61} - 2$ . The number of distinct streams available is roughly  $2^{58}$ .

This generator only accept the default parameter thus 0 is the only allowed value for para.

#### Value

None.

**Note**

Only one active stream is allowed for each R process. Multiple streams per process can be achieved by saving (packing) the states of the streams and unpacking when needed.

**Author(s)**

Na (Michael) Li <nali@umn.edu>

**References**

SPRNG: Scalable Parallel Random Number Generator Library Web Page. <http://sprng.cs.fsu.edu/>

**See Also**

[free.sprng](#), [pack.sprng](#), [spawn.sprng](#), [spawn.new.sprng](#)

**Examples**

```
init.sprng (1, 0, kind = "MLFG", para = 6)
runif (10)
free.sprng ()
```

---

pack.sprng	<i>Return the PRNG state.</i>
------------	-------------------------------

---

**Description**

pack.sprng returns the state of current active PRNG stream as a vector of integers.

**Usage**

```
pack.sprng ()
```

**Value**

Return the current PRNG state as an integer vector or NULL if there is no active PRNG stream.

**Author(s)**

Na (Michael) Li <nali@umn.edu>

**References**

SPRNG: Scalable Parallel Random Number Generator Library Web Page. <http://sprng.cs.fsu.edu/>

**See Also**

[unpack.sprng](#), [spawn.sprng](#)

**Examples**

```
init.sprng (1, 0, kind = "LCG")
runif (10)
saved <- pack.sprng ()
a1 <- runif (10)
unpack.sprng (saved)
a2 <- runif (10)
all (a1 == a2)
free.sprng ()
```

---

spawn.new.sprng	<i>Spawn several new random number streams.</i>
-----------------	---

---

**Description**

spawn.new.sprng generates several new PRNG streams at one pass.

**Usage**

```
spawn.new.sprng (nstream, seed = 0, kindprng = "default", para = 0)
```

**Arguments**

nstream	number of new streams to spawn
seed	an integer of random number seed. It is not the starting state of the sequence; rather, it is an encoding of the starting state. The same seed for all the streams. Distinct streams are returned. Only the 31 least significant bits of seed are used in determining the initial starting state of the stream. Default value is 0 where the default seed for each random number generator is used
kindprng	a character string of the disired kind of parallel random number generator
para	additional parameters for the parallel random number generators. If para is 0, default parameters for each PRNG are used. When invalid parameter is given, a warning is issued and the default paramter is used.

**Details**

Usually one would call [init.sprng](#) separately for each process or use [spawn.sprng](#) to spawn new streams from current stream. spawn.new.sprng generates new streams unrelated to the current one (might be even of different type). Unlike [init.sprng](#), [RNGkind](#) is *not* called.

**Value**

Return an integer matrix with nstream columns, each column corresponds to one stream. The number of rows will depend on the type of the PRNG.

**Author(s)**

Na (Michael) Li <nali@umn.edu>

**References**

SPRNG: Scalable Parallel Random Number Generator Library Web Page. <http://sprng.cs.fsu.edu/>

**See Also**

[pack.sprng](#), [unpack.sprng](#), [spawn.sprng](#)

**Examples**

```
prngs <- spawn.new.sprng (3, kind = "MLFG")
unpack.sprng (prngs[,2])
RNGkind ("user")
runif (10)
free.sprng ()
```

---

spawn.sprng

*Spawn new random number streams based on the current one.*

---

**Description**

Some times when a process spawns children processes, it is desirable to spawn new streams from an old one and pass them on to children. `spawn.sprng` creates new random number streams based on the current one, and returns them as columns of an integer matrix. These new streams can be then transfered to children processes.

**Usage**

```
spawn.sprng (nspawn)
```

**Arguments**

nspawn            number of new streams to spawn

**Value**

Return an integer matrix with nstream columns, each column corresponds to one stream. The number of rows will depend on the type of the PRNG.

**Author(s)**

Na (Michael) Li <nali@umn.edu>

**References**

SPRNG: Scalable Parallel Random Number Generator Library Web Page. <http://sprng.cs.fsu.edu/>

**See Also**

[pack.sprng](#), [unpack.sprng](#), [spawn.new.sprng](#)

**Examples**

```
init.sprng (1, 0, kind = "CMRG")
child.rngs <- spawn.sprng (3)
unpack.sprng (child.rngs[,1])
runif (10)
unpack.sprng (child.rngs[,2])
runif (10)
free.sprng ()
```

---

type.sprng

*Return the type of the active SPRNG stream*

---

**Description**

type.sprng returns the type of current active SPRNG stream.

**Usage**

```
type.sprng ()
```

**Value**

Return the a string, the type of active SPRNG.

**Author(s)**

Na (Michael) Li <nali@umn.edu>

**References**

SPRNG: Scalable Parallel Random Number Generator Library Web Page. <http://sprng.cs.fsu.edu/>

**See Also**

[init.sprng](#),

## Examples

```
init.sprng (1, 0, kind = "MLFG", para = 6)
type.sprng ()
free.sprng ()
```

---

unpack.sprng	<i>Restore the PRNG state.</i>
--------------	--------------------------------

---

## Description

unpack.sprng unpacks a saved PRNG state and replace the current active PRNG state with the new one.

## Usage

```
unpack.sprng (rngstate)
```

## Arguments

rngstate            an integer vector representing PRNG state (should be returned by either [pack.sprng](#), [spawn.sprng](#), [spawn.new.sprng](#))

## Value

Return the states of the old PRNG stream if there is one and return NULL if not.

## Author(s)

Na (Michael) Li <nal@umn.edu>

## References

SPRNG: Scalable Parallel Random Number Generator Library Web Page. <http://sprng.cs.fsu.edu/>

## See Also

[pack.sprng](#), [spawn.sprng](#), [spawn.new.sprng](#)

## Examples

```
init.sprng (1, 0, seed = 1132, kind = "LCG64")
runif (5)
rng.saved <- pack.sprng ()
runif (5)
unpack.sprng (rng.saved)
runif (5)
free.sprng ()
```

# Index

## \*Topic **distribution**

- free.sprng, [2](#)
- init.sprng, [3](#)
- pack.sprng, [6](#)
- spawn.new.sprng, [7](#)
- spawn.sprng, [8](#)
- type.sprng, [9](#)
- unpack.sprng, [10](#)

## \*Topic **interface**

- free.sprng, [2](#)
- init.sprng, [3](#)
- pack.sprng, [6](#)
- spawn.new.sprng, [7](#)
- spawn.sprng, [8](#)
- type.sprng, [9](#)
- unpack.sprng, [10](#)

free.sprng, [2](#), [6](#)

init.sprng, [2](#), [3](#), [7](#), [9](#)

pack.sprng, [2](#), [6](#), [6](#), [8–10](#)

RNGkind, [2](#), [7](#)

spawn.new.sprng, [6](#), [7](#), [9](#), [10](#)

spawn.sprng, [6](#), [7](#), [8](#), [8](#), [10](#)

type.sprng, [9](#)

unpack.sprng, [7–9](#), [10](#)